strncpy() and strncat()

Daniel Plakosh, Software Engineering Institute [vita[1]]

2005-09-27

The standard C library includes functions that are designed to prevent buffer overflows, particularly `strncpy()` and `strncat()`. These universally available functions discard data larger than the specified length, regardless of whether it fits into the buffer. These functions are deprecated for new Windows code because they are frequently used incorrectly.

Development Context

Copying and concatenating character strings

Technology Context

C, UNIX, Win32

Attacks

Attacker executes arbitrary code on machine with permissions of compromised process or changes the behavior of the program.

Risk

Improper use of the `strncpy()` and `strncat()` functions can result in buffer overflow vulnerabilities.

Description

The standard C library includes functions that are designed to prevent buffer overflows, particularly `strncpy()` and `strncat()`. These universally available functions discard data larger than the specified length, regardless of whether it fits into the buffer. These functions are deprecated for new Windows code because they are frequently used incorrectly.

The `strncpy()` library function performs a similar function to `strcpy()` but allows a maximum size to be specified:

```
strncpy(dest, source, dest_size - 1);
dest[dest_size - 1] = ’\0’;
```

The `strcat()` function concatenates a string to the end of a buffer. Like `strcpy()`, `strcat()` has a more secure version, `strncat()`. Functions like `strncpy()` and `strncat()` restrict the number

[1] daisy:268 (Plakosh, Daniel)

of bytes written and are generally more secure, but they are not foolproof. The following is an actual example of code that can result from a simplistic transformation of existing code:

```
strncpy(record, user, MAX_STRING_LEN - 1);
strncat(record, cpw, MAX_STRING_LEN - 1);
```

The problem is that the last argument to `strncat()` should not be the total buffer length, it should be the space remaining after the call to `strncpy()`. Both functions require that you specify the remaining space and not the total size of the buffer. Because the remaining space changes every time data is added or removed, programmers must track or constantly recompute the remaining space. These processes are error prone and can lead to vulnerabilities, but the following call correctly calculates the remaining space when concatenating a string using `strncat()`:

```
strncat(dest, source, dest_size-strlen(dest)-1);
```

Another problem with `strncpy()` and `strncat()` is that neither function provides a status code or reports when the resulting string is truncated. Both functions return a pointer to the destination buffer, requiring significant effort by the programmer to determine whether the resulting string was truncated.

The `strncpy()` function doesn't null terminate the destination string if the source string is at least as long as the destination. As a result, the destination string must be null terminated after calling `strncpy()`. In certain circumstances, a failure to null-terminate could lead to a buffer overflow vulnerability.

There's also a performance problem with `strncpy()` in that it fills the entire destination buffer with null bytes after the source data has been exhausted. Although there is no good reason for this behavior, programs now depend on it and it is difficult to change.

References

[ISO/IEC 99]	ISO/IEC. *ISO/IEC 9899 Second edition 1999-12-01 Programming languages — C*. International Organization for Standardization, 1999.

Pearson Education, Inc. Copyright

Velden

Naam	Waarde
Copyright Holder	Pearson Education

Velden

Naam	Waarde
is-content-area-overview	false
Content Areas	Knowledge/Coding Practices
SDLC Relevance	Implementation
Workflow State	Publishable